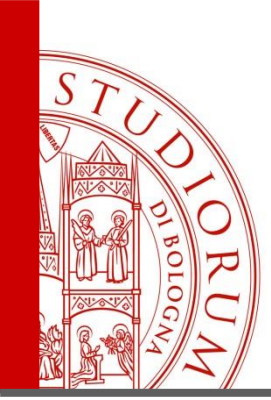


Università degli Studi di Bologna
Corso di Laurea in Ingegneria Informatica

Progettazione per la Sicurezza

Ingegneria del Software T

Prof. MARCO PATELLA
Dipartimento di Informatica – Scienza e Ingegneria (DISI)

Sommario

- Progettazione per la sicurezza
- Testare la sicurezza
- Capacità di sopravvivenza del sistema
- Conclusioni

PROGETTAZIONE PER LA SICUREZZA

Progettazione per la Sicurezza

- **La sicurezza non è qualcosa che può essere aggiunto al sistema**
- La sicurezza ***deve essere progettata insieme al sistema*** prima dell'implementazione
- "Sicurezza" **è anche un problema implementativo**: spesso le vulnerabilità sono introdotte durante la fase di implementazione
  - È possibile ottenere un'implementazione non sicura da una progettazione sicura
  - Non è possibile ottenere un'implementazione sicura partendo da una progettazione non sicura

Progettazione Architetturale

- La scelta dell'architettura del sistema influenza profondamente la sicurezza
- Un'architettura inappropriata non garantisce
  - riservatezza ed integrità delle informazioni
  - il livello di disponibilità richiesto
- Due problemi fondamentali vanno considerati quando si progetta l'architettura del sistema:
  - *Protezione*: come dovrebbe essere organizzato il sistema in modo che i beni critici possano essere protetti dagli attacchi esterni?
  - *Distribuzione*: come dovrebbero essere distribuiti i beni in modo da minimizzare gli effetti di un attacco andato a buon fine?

Progettazione Architetturale

- I due problemi sono potenzialmente in conflitto:
  - se si mettono tutti i beni in un unico posto si può costruire un buon livello di protezione a un costo non eccessivo
  - se però la protezione fallisce tutti i beni sono compromessi
  - distribuire i beni porta ad un maggiore costo per la protezione
  - ci sono più possibilità che la protezione possa fallire se i beni sono distribuiti, ma se questo avviene vi è la perdita solo parziale dei beni

Progettazione Architetturale

- Tipicamente la migliore architettura per fornire un alto grado di protezione è quella a layer
- I beni critici da proteggere sono posizionati al livello più basso
- Il numero di layer necessari varia da applicazione ad applicazione e dipende dalla criticità dei beni che devono essere protetti
- Per migliorare la protezione inoltre sarebbe bene che le credenziali di accesso ai diversi livelli fossero diverse tra loro
  - Esempio: se si adotta un meccanismo di accesso con password, ogni livello deve avere una propria password diversa da quelle degli altri livelli

Esempio

Protezione a livello di piattaforma

- Autenticazione di sistema
- Autorizzazione di sistema
- Gestione integrità file

Protezione a livello di applicazione

- Login al database
- Autorizzazione di database
- Gestione della transazione
- Ripristino del database

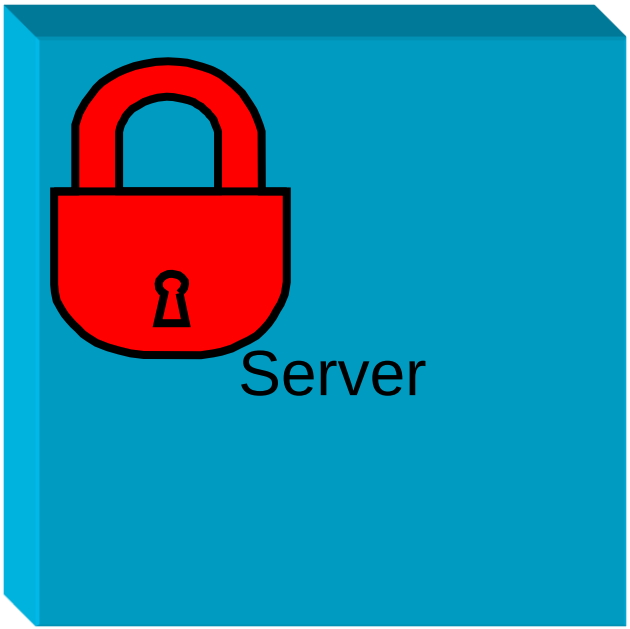
Protezione a livello di record

- Autorizzazione accesso record
- Cifratura record
- Gestione integrità record

Dati dei pazienti

Progettazione Architetturale

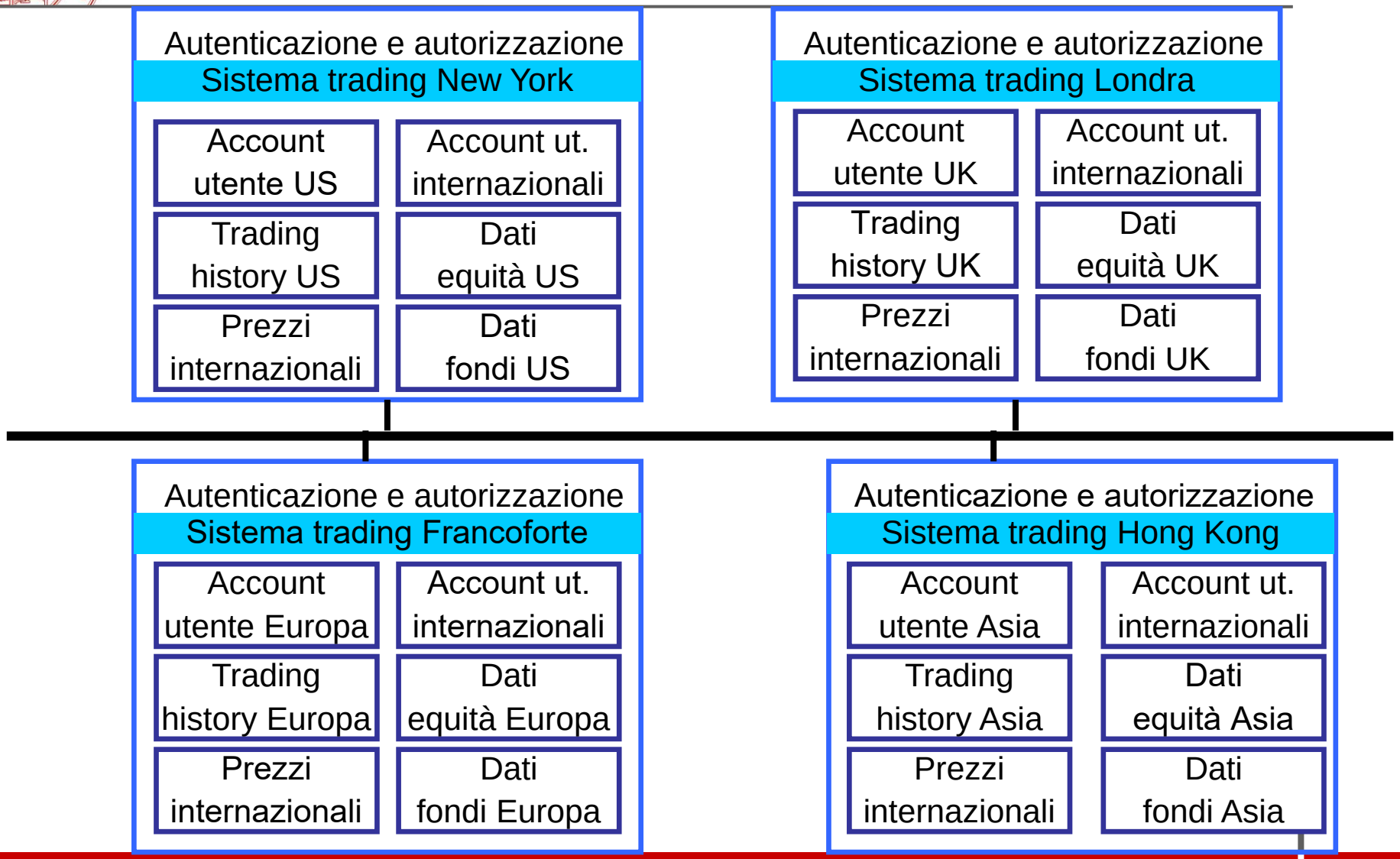
- Se la protezione dei dati è un requisito critico
  si potrebbe anche usare un'architettura client-server
  con i meccanismi di protezione nella macchina server



Progettazione Architetturale

- La versione tradizionale client-server ha molte limitazioni
- Se la sicurezza viene compromessa
  - le perdite associate ad un attacco saranno alte
    Esempio: tutte le credenziali di accesso
    verranno compromesse
  - i costi di recupero saranno anch'essi elevati
    Esempio: tutte le credenziali di accesso al sistema
    andranno rigenerate
- Il sistema è inoltre maggiormente soggetto ad attacchi DoS che sovraccaricano il server
- Una possibile soluzione può essere quella
  di adottare una architettura distribuita
  in cui il server viene replicato in punti diversi della rete

Esempio


Autenticazione e autorizzazione
Sistema trading New York
Account utente US
Account ut. internazionali
Trading history US
Dati equità US
Prezzi internazionali
Dati fondi US
Autenticazione e autorizzazione
Sistema trading Londra
Account utente UK
Account ut. internazionali
Trading history UK
Dati equità UK
Prezzi internazionali
Dati fondi UK
Autenticazione e autorizzazione
Sistema trading Francoforte
Account utente Europa
Account ut. internazionali
Trading history Europa
Dati equità Europa
Prezzi internazionali
Dati fondi Europa
Autenticazione e autorizzazione
Sistema trading Hong Kong
Account utente Asia
Account ut. internazionali
Trading history Asia
Dati equità Asia
Prezzi internazionali
Dati fondi Asia

Esempio

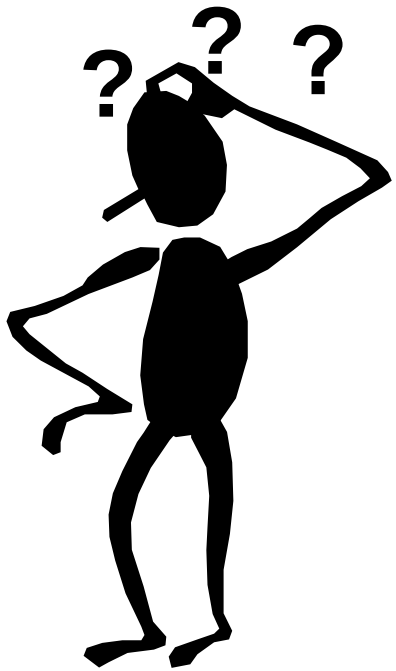
- I beni del sistema sono distribuiti in diversi nodi della rete ognuno dei quali ha un proprio meccanismo di protezione dei dati
- I dati “più importanti” sono replicati nei diversi nodi
- Attacco ad uno specifico nodo:
  - alcuni beni non saranno disponibili
  - il sistema comunque potrà ancora funzionare e fornire i servizi più importanti
  - grazie alla replicazione, il ripristino dei dati nel nodo attaccato sarà più facile e meno costoso

Progettazione Architetturale

- Tipico problema: lo stile architetturale più appropriato per la sicurezza potrebbe essere in conflitto con gli altri requisiti dell'applicazione (analisi trade-off)
- Esempio:
  a. riservatezza dei dati memorizzati su un vasto database
  b. accesso molto veloce ai dati
- Soddisfare entrambi i requisiti nella stessa architettura presenta molti problemi
  a. → layer per garantire la riservatezza
     → diminuzione netta della velocità di accesso ai dati
  b. → architettura "snella"
     → diminuzione netta della riservatezza

Progettazione Architetturale

- Tipico problema: lo stile architetturale più appropriato per la sicurezza potrebbe essere in conflitto con gli altri requisiti dell'applicazione (analisi trade-off)
- Va valutato attentamente quale «requisito» è prioritario e l'architettura sarà scelta di conseguenza



Linee Guida di Progettazione

- Non ci sono regole rigide per ottenere un sistema sicuro
- ***Differenti tipi di sistema richiedono differenti misure tecniche per ottenere un livello di sicurezza accettabile***
- La posizione e i requisiti di diversi gruppi di utenti influenzano pesantemente **cosa è** e **cosa non è** accettabile
- Ci sono comunque linee guida generali di ampia applicabilità per la progettazione di sistemi sicuri che possono fungere da:
  - mezzo per migliorare la consapevolezza dei problemi di sicurezza in un team di progettisti software
  - base per una lista di controlli da fare durante il processo di validazione del sistema

Linee Guida di Progettazione

1. Basare le decisioni della sicurezza su una politica esplicita
2. Evitare un singolo punto si fallimento
3. Fallire in modo certo
4. Bilanciare sicurezza e usabilità
5. Essere consapevoli dell'esistenza dell'ingegneria sociale
6. Usare ridondanza e diversità riduce i rischi
7. Validare tutti gli input
8. Dividere in compartimenti i beni
9. Progettare per il deployment
10. Progettare per il ripristino

Basare la Sicurezza su Policy

- La ***Security Policy*** è un documento di alto livello che definisce “cosa” è la sicurezza ma non “come” ottenerla
- La policy non dovrebbe definire i meccanismi usati per fornire e far rispettare la sicurezza
- Gli aspetti della security policy dovrebbero originare dei requisiti di sistema
- In pratica ciò è poco probabile, specie se viene adottato un processo di sviluppo rapido
- I progettisti dovrebbero quindi consultare la policy sia nelle decisioni di progettazione che nella loro valutazione

Progettazione delle Politiche

- Le politiche di sicurezza devono essere incorporate nella progettazione al fine di:
  - specificare come le informazioni possono essere accedute
  - quali precondizioni devono essere testate per l'accesso
  - a chi concedere l'accesso
- Tipicamente le politiche vengono rappresentate come un insieme di regole e condizioni
- Tali regole devono essere incorporate in uno specifico componente del sistema chiamato "***Security Authority***" che avrà il compito di far rispettare le politiche all'interno dell'applicazione

Progettazione delle Politiche

- A livello progettuale le politiche di sicurezza sono suddivise in sei specifiche categorie:
  - *Identity policies*: definiscono le regole per la verifica delle credenziali degli utenti
  - *Access control policies*: definiscono le regole da applicare sia alle richieste di accesso alle risorse sia all'esecuzione di specifiche operazioni messe a disposizione dall'applicazione
  - *Content-specific policies*: definiscono le regole da applicare a specifiche informazioni durante la memorizzazione e la comunicazione

Progettazione delle Politiche

- *Network and infrastructure policies:* definiscono le regole per controllare il flusso dei dati e il deployment sia delle reti che dei servizi infrastrutturali di hosting pubblici e privati
- *Regulatory policies:* definiscono le regole a cui l'applicazione deve sottostare per soddisfare i requisiti legali e di regolamentazione delle leggi in vigore nel Paese/Stato in cui il sistema opera
- *Advisor and information policies:* queste regole non sono imposte, ma sono caldamente consigliate in riferimento alle regole dell'organizzazione e al ruolo delle attività di business.
  Per esempio queste regole possono essere applicate per informare il personale sull'accesso ai dati sensibili o per stabilire comunicazioni commerciali con partner esterni

Evitare Punto Singolo di Fallimento

- Nei sistemi critici è buona norma di progettazione quella di cercare di **evitare un singolo punto di fallimento**
- Questo perché un singolo fallimento in una parte del sistema non si trasformi nel fallimento di tutto il sistema
- Per quanto riguarda la sicurezza questo significa che **non ci si dovrebbe affidare a un singolo meccanismo per assicurarla**, ma si dovrebbero impiegare differenti tecniche
- Questo viene spesso chiamato "***difesa in profondità***"
- Esempio: se si usa la password per autenticare si dovrebbe anche includere un meccanismo sfida e risposta

Fallire in Modo Certo

- Qualche tipo di fallimento è inevitabile in tutti i sistemi, ma i sistemi critici per la sicurezza dovrebbero sempre ***fallire in modo sicuro***
- Non si dovrebbero avere procedure di fall-back meno sicure del sistema stesso
- Anche se il sistema fallisce **non deve essere consentito a un attaccante di accedere ai dati riservati**
- Esempio: i dati dei pazienti dovrebbero essere scaricati sul client all'inizio di ogni sessione clinica, così se il server fallisce i dati sono comunque mantenuti sul client. I dati vengono cifrati per non essere letti da personale non autorizzato

Bilanciare Sicurezza e Usabilità

- Sicurezza e usabilità sono spesso in contrasto
  - per avere sicurezza bisogna introdurre un numero di controlli che garantiscano che gli utenti siano autorizzati a usare il sistema e che nello stesso tempo agiscano in accordo alle politiche di sicurezza
  - questo inevitabilmente ricade sull'utente che ha bisogno di più tempo per imparare ad utilizzare il sistema
- Ogni volta che si aggiunge una caratteristica di sicurezza al sistema questo inevitabilmente diventa meno usabile
- A volte può diventare contro produttivo introdurre nuove caratteristiche di sicurezza a spese dell'usabilità
  - Esempio: obbligare l'utente all'adozione di password forti

Essere Consapevoli dell'Ingegneria Sociale

- ***Ingegneria sociale***: trovare modi per convincere con l'inganno utenti accreditati al sistema a rivelare informazioni riservate
- Questi approcci si avvantaggiano della "**volontà di aiutare**" delle persone e della loro fiducia nell'organizzazione
- Dal punto di vista della progettazione contrastare l'ingegneria sociale **è quasi impossibile**
- Se la sicurezza è molto critica non si dovrebbe affidarsi solo a meccanismi di autenticazione basati su password, ma bisognerebbe utilizzare meccanismi di autenticazione forte
- Meccanismi di log che tracciano sia la locazione che l'identità dell'utente e programmi di analisi del log potrebbero essere utili ad identificare brecce nella sicurezza

Usare Ridondanza e Diversità

- Ridondanza significa **mantenere più di una versione** del software e dei dati nel sistema
- Diversità significa **che le diverse versioni del sistema non dovrebbero usare la stessa piattaforma o essere basati sulle stesse tecnologie**
- In questo modo una vulnerabilità della piattaforma o della tecnologia non influirà su tutte le versioni e non condurrà a un comune punto di fallimento
- Esempio:
  - mantenere i dati dei pazienti sia sul client che sul server
  - client e server devono avere un diverso sistema operativo
  - attacco basato su vulnerabilità del SO non influenza sia client che server

Validare tutti gli Input

- Un comune attacco al sistema consiste nel fornire input inaspettati che causano un comportamento imprevisto
  - crash
  - perdita della disponibilità del servizio
  - esecuzione di codice malevolo
- Tipici esempi sono buffer overflow e SQL injection
- Si possono evitare molti di questi problemi progettando la validazione dell'input in tutto il sistema
- Nei requisiti dovrebbero essere definiti tutti i controlli che devono essere applicati
- Bisogna usare la conoscenza dell'input per definire questi controlli

Dividere in Compartimenti i Beni

- *Compartimentalizzare* significa organizzare le informazioni nel sistema in modo che gli ***utenti abbiano accesso solo alle informazioni necessarie*** piuttosto che a tutte le informazioni del sistema
- Gli effetti di un attacco in questo modo sono più contenuti: qualche informazione sarà persa o danneggiata, ma è poco probabile che tutte le informazioni del sistema siano coinvolte
- Esempio:
  - lo staff clinico può avere accesso soltanto ai record dei pazienti che hanno un appuntamento o sono ricoverati nella clinica
  - esiste un meccanismo per gestire accessi inaspettati

Progettazione per il Deployment

- Molti problemi di sicurezza sorgono perché il sistema ***non viene configurato correttamente*** al momento del deployment
- Bisogna sempre progettare il sistema in modo che
  - siano inclusi programmi di utilità per semplificare il deployment
  - verificare potenziali errori di configurazione e omissioni nel sistema di deployment

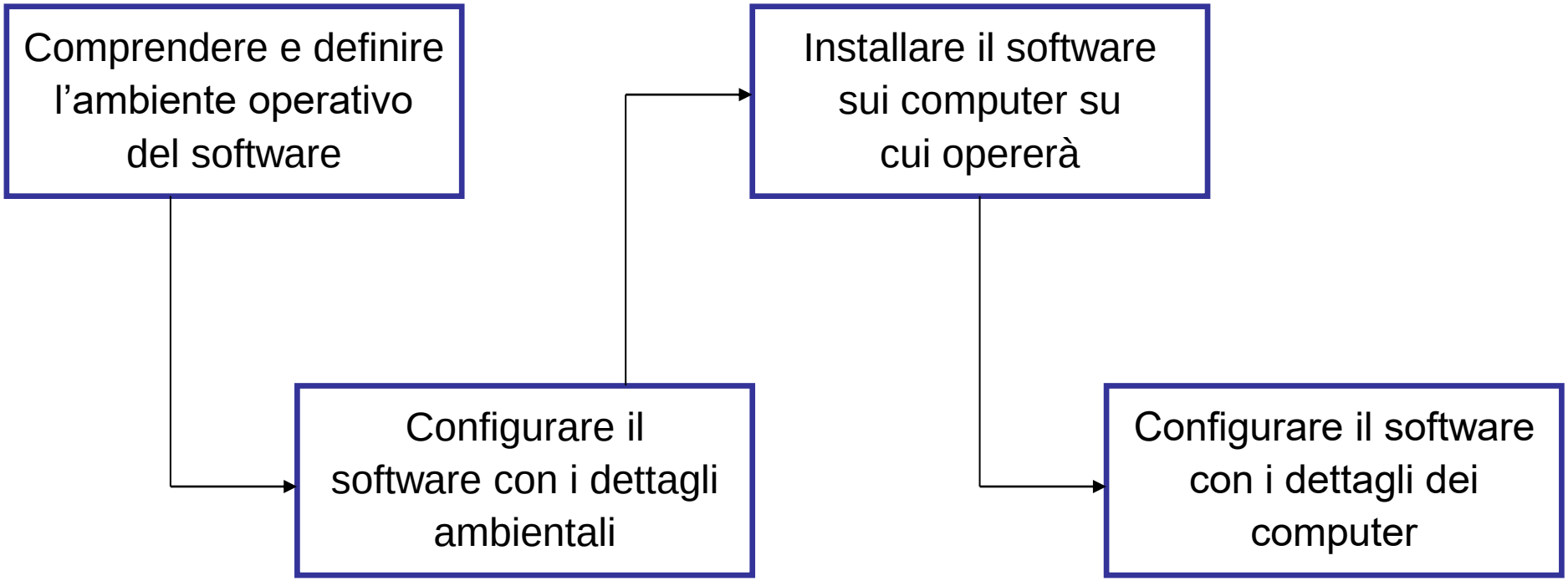
Progettazione per il Ripristino

- Bisogna sempre progettare il sistema con l'assunzione che gli errori di sicurezza possano accadere
- Si deve quindi pensare a come ripristinare il sistema dopo possibili errori e riportarlo a uno stato operazionale sicuro
- Esempio:
  - persone non autorizzate accedono ai dati dei pazienti
  - non è noto come abbiano ottenuto le credenziali per l'accesso
  - occorre quindi cambiare tutte le credenziali del sistema in modo che la persona non autorizzata non abbia accesso al meccanismo di cambiamento delle password

Progettazione per il Deployment

- Il deployment di un sistema coinvolge:
  - configurazione del sistema per operare nell'ambiente:
    - semplice impostazione di parametri delle preferenze degli utenti
    - definizione di regole e modelli di business che governano l'esecuzione del software
  - installazione del sistema sui computer dell'ambiente
  - configurazione del sistema installato

Deployment del Software



Progettazione per il Deployment

- Nella fase di deployment vengono spesso introdotte in modo accidentale delle vulnerabilità
- Esempio:
  - il software deve spesso essere configurato con una lista di utenti autorizzati
  - quando il software è rilasciato questa lista consiste di un login per l'amministratore generico come "admin" e la password di default è "password"
  - come prima azione l'amministratore dovrebbe modificare i dati di login, ma è molto facile dimenticare di farlo
  - un attaccante che conosce il login di default potrebbe essere capace di guadagnare privilegi di accesso al sistema

Progettazione per il Deployment

- La configurazione e il deployment sono spesso visti solo come problemi di amministrazione e quindi al di fuori del processo di ingegnerizzazione
- I progettisti software hanno la responsabilità di ***progettare per il deployment***
- Bisogna sempre fornire supporti per il deployment che riducano la probabilità che gli amministratori compiano degli errori quando configurano il software
- Esistono delle linee guida per la progettazione per il deployment

Linee Guida

1. Includere supporto per visionare
   e analizzare le configurazioni
2. Minimizzare i privilegi di default
3. Localizzare le impostazioni di configurazione
4. Fornire modi per rimediare a vulnerabilità di sicurezza

Supporto per le Configurazioni

- Si devono sempre includere **programmi di utilità** che consentano agli amministratori di esaminare la configurazione corrente del sistema
- Sorprendentemente questi programmi **mancano** nella maggior parte dei sistemi software
- Gli utenti sono spesso frustrati dalla difficoltà di trovare i dettagli della configurazione
  - per un quadro completo della configurazione spesso occorre visionare diversi menu e questo porta a errori e omissioni
- Idealmente in fase di visualizzazione delle configurazioni si dovrebbero evidenziare impostazioni critiche per la sicurezza

Minimizzare i Privilegi di Default

- Il software deve essere progettato in modo tale che la ***configurazione di default fornisca i minimi privilegi essenziali***
- In questo modo vengono limitati i danni di un possibile attacco
- Esempio:
  - l'autenticazione di default dell'amministratore dovrebbe solo consentire l'accesso a un modulo che permette all'amministratore di inserire nuove credenziali
  - non dovrebbe essere consentito l'accesso a nessuna altra funzionalità
  - quando vengono modificate le credenziali, quella di default dovrebbe essere automaticamente cancellata

Localizzare le Impostazioni di Configurazione

- Quando si progetta il supporto per le configurazioni del sistema bisognerebbe assicurarsi che ogni risorsa che appartiene alla stessa parte del sistema venga configurata nella stessa posizione
- Se le informazioni di configurazione non sono localizzate
  - è facile dimenticarsi di farlo
  - può capitare di non essere a conoscenza dell'esistenza di meccanismi per la sicurezza già inclusi nel sistema
  - se tali meccanismi presentano configurazioni di default si potrebbe essere esposti ad attacchi

Rimediare a Vulnerabilità

- Bisogna includere **meccanismi diretti** per:
  - aggiornare il sistema
  - riparare le vulnerabilità di sicurezza che vengono scoperte
- Questi potrebbero includere
  - verifiche automatiche per aggiornamenti di sicurezza
  - download di tali aggiornamenti non appena sono disponibili
- Va comunque considerato che gli aggiornamenti devono coinvolgere contemporaneamente centinaia di PC su cui tipicamente il software è installato

TESTARE LA SICUREZZA

Testare la Sicurezza

- Il test di un sistema gioca **un ruolo chiave** nel processo di sviluppo software e dovrebbe essere eseguito con molta attenzione
- È quindi sorprendente che **l'area dei test della sicurezza sia quella più trascurata durante lo sviluppo del sistema**
- Questo può essere attribuito a diversi fattori:
  - mancanza di comprensione dell'importanza dei test relativi alla sicurezza
  - mancanza di tempo
  - mancanza di conoscenza su come svolgere un test di sicurezza
  - mancanza di tool integrati per compiere test

Testare la Sicurezza

- Il test della sicurezza è un lavoro molto lungo e tedioso, spesso molto più complesso dei test funzionali che vengono svolti normalmente
- Inoltre esso coinvolge diverse discipline
  - ci sono tradizionali test per accertare la sicurezza dei requisiti applicativi che possono essere svolti normalmente dal team di testing
  - ma esistono dei test non funzionali di “rottura” del sistema che devono essere condotti da esperti di sicurezza
    - Black box testing
    - White box testing

Black Box Testing

- Questo test ha come assunzione di base la non conoscenza dell'applicazione
- I tester affrontano l'applicazione come farebbe un attaccante
  - indagando sulle informazioni riguardanti la struttura interna
  - successivamente applicano un insieme di tentativi di violazione del sistema basati sulle informazioni ottenute
- Esempio: se un URL di una applicazione contiene una estensione ".cgi" allora può essere inferito che l'applicazione è stata sviluppata con la tecnologia CGI e applicare quindi le ben conosciute tecniche di violazione di questa tecnologia

Black Box Testing

- I tester possono impiegare una varietà di strumenti per scansionare e indagare l'applicazione
  - ci sono centinaia di tool in rete per l'hacking di applicazioni che permettono di "scandagliare" le porte dei sistemi perpetuando attacchi sfruttando le debolezze ben conosciute di svariati linguaggi di programmazione
- Questo test non prende in esame solo debolezze del codice, ma vengono svolti test mirati anche al livello infrastrutturale
  - errori di configurazione di reti e host
  - falle di sicurezza nelle macchine virtuali
  - problemi legati ai linguaggi di implementazione

White Box Testing

- Questo test ha come assunzione di base la completa conoscenza dell'applicazione
- I tester hanno accesso a tutte le informazioni di configurazione e anche al codice sorgente
- Essi operano una revisione del codice cercando possibili debolezze
- Inoltre scrivono test per stabilire come trarre vantaggio dalle debolezze scoperte
- Tipicamente questi tester sono ex-sviluppatori o persone che conoscono molto bene l'ambiente di sviluppo

White Box Testing

- I tool a disposizione differiscono molto
  da quelli usati nel black box test
- Tipicamente sono tool di debugging che consentono
  di trovare bachi e vulnerabilità specifici del sistema
- I bachi tipici riguardano problemi di corsa critici
  e la mancanza di verifica dei parametri di input
  e sono specifici di ogni applicazione
- Questi test portano a scoprire anche altri problemi
  come i memory leak e problemi di prestazione
  che contribuiscono al danneggiamento della disponibilità
  e dell'affidabilità dell'intero sistema

CAPACITÀ DI SOPRAVVIVENZA DEL SISTEMA

Capacità di Sopravvivenza

- Con il termine ***capacità di sopravvivenza*** *(survivability)* si intende la capacità del sistema di continuare a fornire i servizi essenziali agli utenti legittimi
  - mentre è sotto attacco
  - dopo che parti del sistema sono state danneggiate come conseguenza di un attacco o di un fallimento
- La capacità di sopravvivenza è una proprietà dell'intero sistema, non dei singoli componenti di questo
- Il lavoro sulla capacità di sopravvivenza è molto critico poiché l'economia e la vita sociale dipendono da infrastrutture controllate da computer

Capacità di Sopravvivenza

- L'analisi e la progettazione della capacità di sopravvivenza dovrebbero essere parte del processo di ingegnerizzazione dei sistemi sicuri
- La disponibilità dei servizi critici è l'essenza della sopravvivenza
- Questo significa conoscere
  - quali sono i servizi maggiormente critici
  - come questi servizi possono essere compromessi
  - qual è la qualità minima dei servizi che deve essere mantenuta
  - come proteggere questi servizi
  - come ripristinare velocemente il sistema se i servizi diventano non disponibili

Esempio

- Un sistema informatico che gestisce l'invio delle ambulanze in risposta alle chiamate di emergenza
- Servizi:
  - prendere le chiamate e inviare le ambulanze
  - log delle chiamate
  - gestione locazione delle ambulanze
- Il servizio critico è quello legato a prendere le chiamate e inviare le ambulanze perché necessita di un processo real-time per la gestione degli eventi

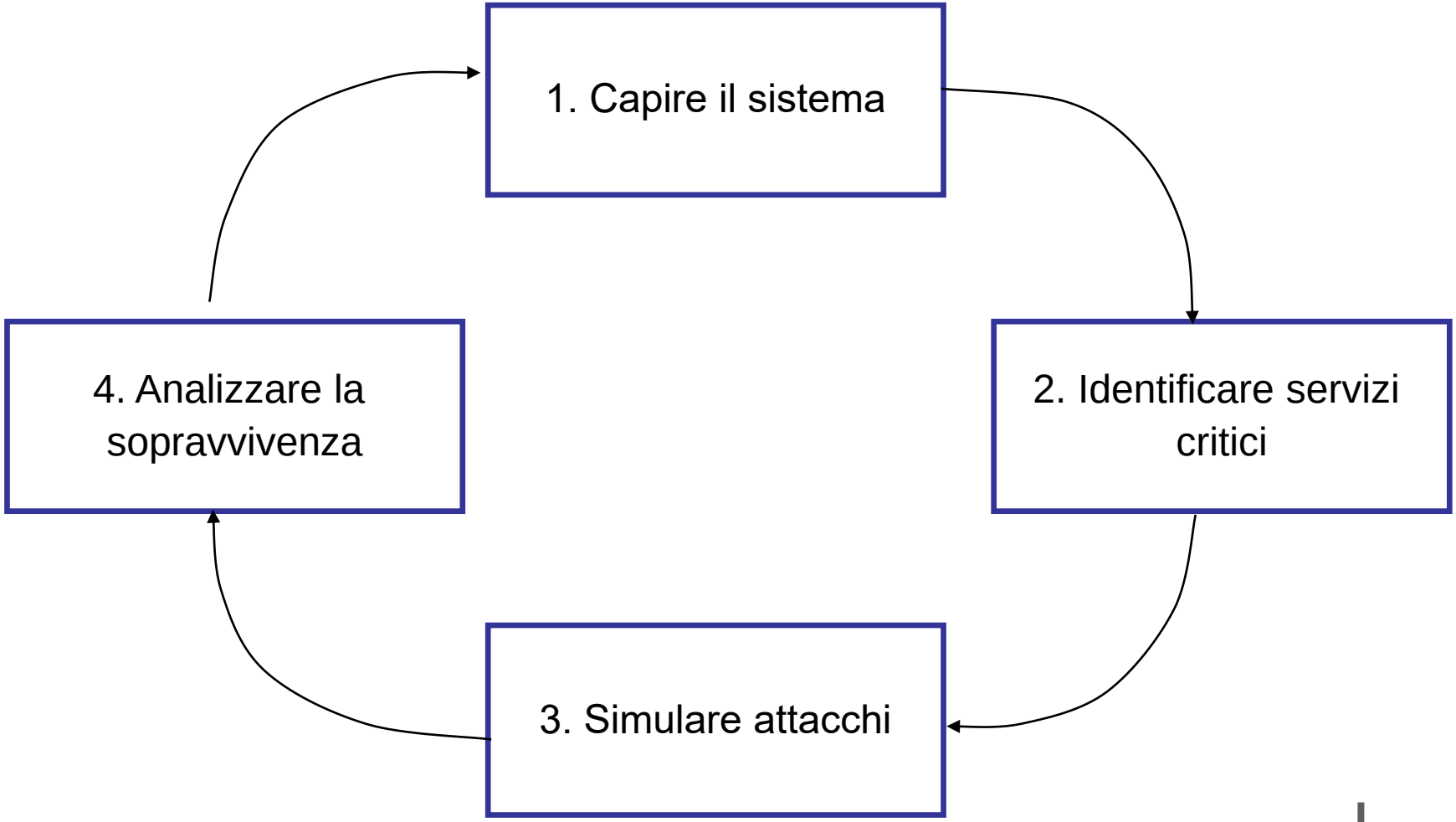
Analisi della Sopravvivenza

- Il Survivable Analysis Systems è un metodo di analisi ideato agli inizi del 2000 per:
  - valutare le vulnerabilità nel sistema
  - supportare la progettazione di architetture e caratteristiche che promuovono la sopravvivenza del sistema
- In questo metodo la sopravvivenza del sistema
  - dipende da tre strategie complementari
  - è un processo a 4 fasi

Strategie

- Resistenza:
  - evitare problemi costruendo all'interno del sistema le capacità di respingere attacchi
  - es: firma digitale per l'autenticazione
- Identificazione:
  - individuare problemi costruendo all'interno del sistema le capacità di riconoscere attacchi e fallimenti e valutare il danno risultante
  - es: aggiungere checksum ai dati critici
- Ripristino
  - tollerare problemi costruendo all'interno del sistema le capacità di fornire servizi essenziali durante un attacco
  - ripristinare le complete funzionalità dopo l'attacco

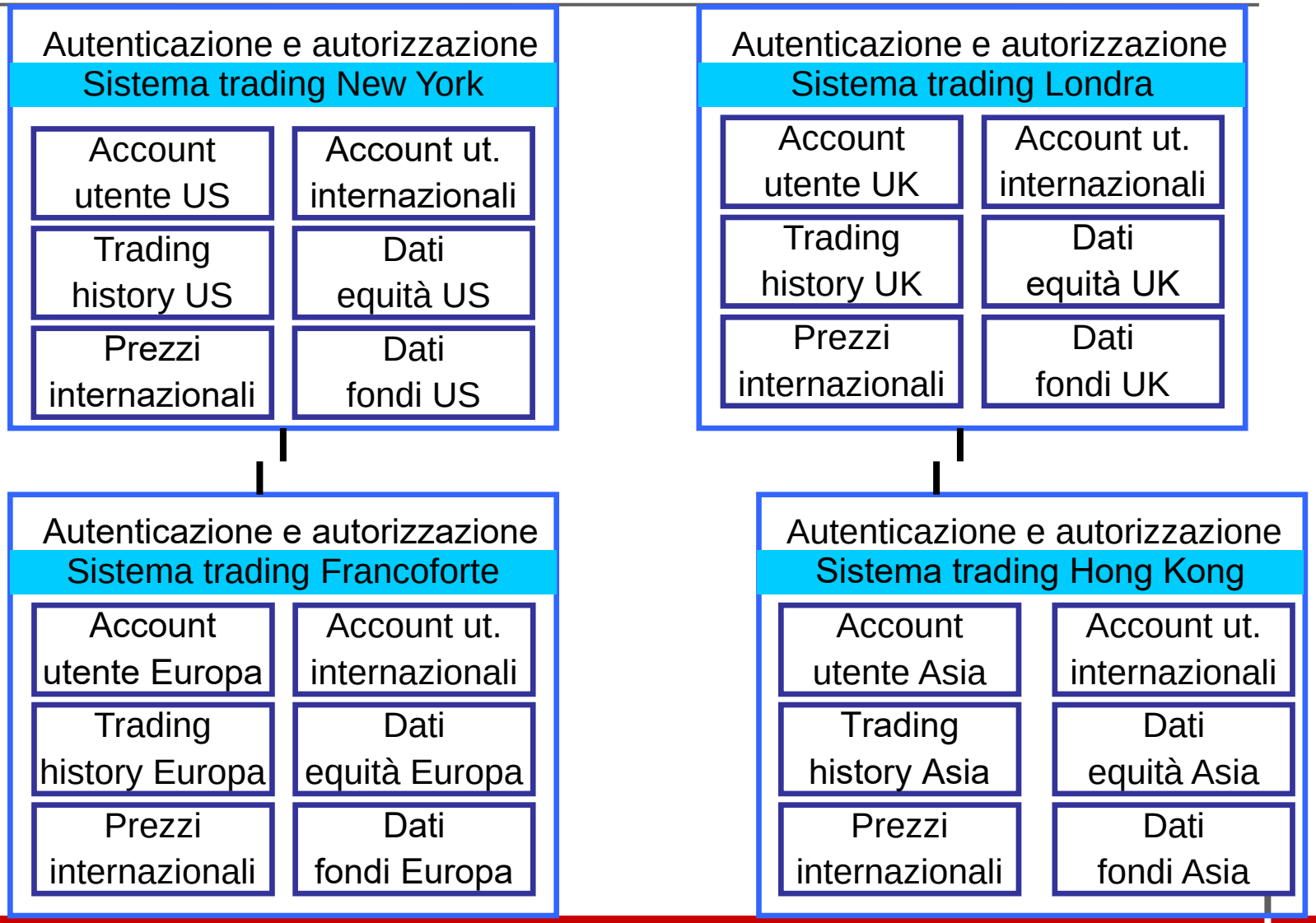
Fasi Analisi di Sopravvivenza



Principali Attività

- *Capire il sistema*: riesaminare gli obiettivi del sistema, i requisiti e l'architettura
- *Identificare servizi critici*: identificare i servizi che devono essere mantenuti e i componenti che devono svolgere tale compito
- *Simulare gli attacchi*: identificare gli scenari o i casi d'uso dei possibili attacchi insieme ai componenti influenzati da questi attacchi
- *Analizzare la sopravvivenza*: identificare
  - i componenti che sono sia essenziali che a rischio
  - le strategie di sopravvivenza basate su resistenza, identificazione e ripristino

Esempio


Autenticazione e autorizzazione
Sistema trading New York
Account utente US
Account ut. internazionali
Trading history US
Dati equità US
Prezzi internazionali
Dati fondi US
Autenticazione e autorizzazione
Sistema trading Londra
Account utente UK
Account ut. internazionali
Trading history UK
Dati equità UK
Prezzi internazionali
Dati fondi UK
Autenticazione e autorizzazione
Sistema trading Francoforte
Account utente Europa
Account ut. internazionali
Trading history Europa
Dati equità Europa
Prezzi internazionali
Dati fondi Europa
Autenticazione e autorizzazione
Sistema trading Hong Kong
Account utente Asia
Account ut. internazionali
Trading history Asia
Dati equità Asia
Prezzi internazionali
Dati fondi Asia

Esempio

- Sistema che gestisce l'equità dei prezzi nei mercati internazionali
- Qual è il minimo supporto che il sistema fornisce già per la sopravvivenza?
  - gli account dei clienti e i prezzi internazionali sono replicati in tutti i nodi
- Servizio chiave che deve sempre essere mantenuto: capacità di piazzare ordini
  - mantenere l'integrità dei dati
  - ordini accurati e che riflettano le vendite e gli acquisti degli utenti

Esempio

- Per mantenere questo servizio ci sono tre componenti del sistema
  - *autenticazione dell'utente*: consente agli utenti autorizzati di accedere al sistema
  - *quotazione dei prezzi*: consente che la vendita e l'acquisto degli stock dei beni siano quotati
  - *piazzamento ordini*: consente di vendere o comprare beni al dato prezzo di mercato
- Questi componenti fanno uso dei dati
  - relativi agli utenti
  - accedono al database delle transazioni

Esempio

- Possibili attacchi al sistema:
  - utente malevolo guadagna le credenziali di accesso di utente verso cui nutre forte rancore
    - vengono piazzati ordini di vendita e acquisto in modo tali da causare seri problemi all'utente legittimo
  - utente non autorizzato corrompe il database delle transazioni guadagnando permessi per eseguire direttamente query SQL
    - riconciliare gli acquisti e le vendite diventa quindi impossibile

Esempio

Attacco	Resistenza	Identificazione	Ripristino
Ordini malevoli	Usare una password diversa da quella di login per piazzare ordini	Mandare una copia degli ordini per e-mail all'utente autorizzato Mantenere storia degli ordini e controllare pattern inusuali di trading	Fornire meccanismi automatici di "undo" e ripristinare l'account Risarcire l'utente per le perdite subite Assicurarsi contro le perdite
Corruzione database	Gli utenti privilegiati necessitano di meccanismi di autenticazione forte	Mantenere una copia a sola lettura delle transazioni effettuate sui vari nodi su un server internazionale. Controllare periodicamente per individuare corruzioni Mantenere una checksum crittografata di tutte le transazioni	Ripristinare il database con le copie di backup Fornire un meccanismo di ripetizione degli scambi dopo un certo tempo per ricreare il database delle transazioni

Capacità di Sopravvivenza

- Aggiungere le tecniche di sopravvivenza costa soldi
- Spesso le aziende sono molto riluttanti ad investire sulla sopravvivenza, specie se non hanno mai subito attacchi e perdite
- È comunque sempre buona norma investire nella sopravvivenza prima piuttosto che dopo aver già subito un attacco
- L'analisi della sopravvivenza non è ancora inclusa nella maggior parte dei processi di ingegnerizzazione del software
- Con la crescita dei sistemi critici sembra probabile che questo tipo di analisi sarà sempre più utilizzato

CONCLUSIONI

Conclusioni

- La sicurezza deve ***essere onnipresente attraverso tutto il ciclo di sviluppo del software***
- Inoltre la sicurezza deve essere tenuta in considerazione anche attraverso tutti gli strati dell'infrastruttura su cui l'applicazione viene sviluppata
- Per ottenere questo è imperativa l'adozione di un processo che tenga in considerazione le problematiche relative alla sicurezza sin dalle prime fasi dello sviluppo del sistema
- È inoltre necessario che vengano compiuti ***severi test periodici*** a verifica del livello di sicurezza del sistema